SYSTEM AND METHOD FOR REAL-TIME MULTI-DIRECTIONAL FILE-BASED DATA STREAMING EDITOR

FIELD OF THE INVENTION

This invention relates generally to data processing systems and, more specifically, to systems and methods that provide real-time multi-directional file-based data streaming editors.

BENEFIT OF EARLIER FILED APPLICATION

This application claims the benefit of U.S. Provisional Application No. 60/196,088, filed April 11, 2000, U.S. Provisional Application No. 60/223,185, filed August 4, 2000, and U.S. Provisional Application No. 60/251,004, filed December 4, 2000, all of which are incorporated by reference herein.

BACKGROUND OF THE INVENTION

The Internet is increasingly used as a collaboration tool among groups of users who are physically not located in the same area. Collaboration generally includes multi-user access to a file. It is often desirable if the collaborating parties have joint editing and viewing capabilities. For example, remote Internet users may desire to participate in virtual meetings, distance learning, virtual seminars, and online conferencing. The current marketplace offers various systems that allow users to collaborate over a network. However, many of the conventional systems do not provide the safeguards that are useful to protect confidential information on a desktop, while providing complete multi-user access to one or more files.

At a basic level, email and chat systems allow users to exchange information, ideas and files over a network. Both of these systems have obvious shortcomings. An email message is sent from one user to another as a series of packets routed through a network. The packets are not necessarily sent according to the same path and therefore may not arrive at the destination node in order. The packets are re-ordered at the destination node and delivered to the intended recipient once all of the packets have been received. This process may take anywhere from a couple of seconds to several hours. Email is therefore limited by its transmission time and fails to support real-time collaboration. Chat and instant messaging systems provide a more real-time manner for users to exchange text messages. Collaboration may prove more valuable, however, when users are able to exchange and compare files in real-time.

Some systems which allow remote users to exchange and compare files in real-time do exist. These systems can generally be grouped as either desktop sharing or capturing applications, or whiteboard applications. Whiteboard applications provide a common work area where multiple remote users can input data, which is reflected to all other users participating in a session. The users' input is generally in the form of annotations. Desktop sharing and capturing applications allow multiple users to access the contents of an initiating computer, or of an application. Conventional desktop sharing and whiteboard application systems are subject to several shortcomings, some of which are outlined below.

First, these conventional systems generally either (1) provide users with access to all of the files included on the hard drive, in a specific directory of the initiating computer, or relative to a specific application on the initiating computer; (2) or only

provide access to a single file. Additionally, many of the conventional systems fail to provide all of the users with editing, printing, saving, etc. capabilities. Even when all users are provided with complete editing capabilities, the user of the initiating machine generally has superior editing rights that preempt those of other users. This can potentially decrease the value of the collaboration process. Additionally, such systems only allow a single copy of the file (i.e., the copy of the file that is on the initiating machine) to be modified, regardless of which user is initiating the edit operations. Thus, even if a remote user edits the file, only the copy of the file that is located on the initiating computer can be saved. The users must transmit the modified file among themselves via, for example, email, and each user will have to save the modified file outside of the desktop sharing or capturing application. Further, such systems usually limit the file sharing capabilities to a single file. Tools that do support simultaneous access to multiple files may require each user to manually position the file on the screen so that they may be viewed. Still further, users can only view a file that has been opened in application that is supported by their local machines. Finally, conventional systems do not include a control transfer feature that allows one remote user to assign access rights to another remote user. Rather, such systems allow all remote users to access all files of a initiating computer.

Accordingly, a need exists for a tool that will allow multiple remote users to collaborate over a file, while overcoming the shortcomings of conventional systems.

SUMMARY OF THE INVENTION

This invention, in a preferred embodiment, provides an online, multi-directional file-based data streaming editor that allows multiple users to simultaneously compare, merge, and instantly edit files, while concurrently communicating with one another over a dedicated connection.

In accordance with an embodiment of the invention, an online file editing method is provided. The method includes creating a session that allows at least two users with simultaneous to access a file, where each user has access to a replica of the file, receiving from a user an edit instruction indicating a file edit, editing a replica of the file according to the edit instruction, and automatically cascading the file edit to each replica of the file.

In accordance with another embodiment of the invention, an online file editing method is provided. The method includes providing at least two users with simultaneous access to a file such that each user has access to a replica of the file, providing one of the at least two users with a capability to edit a local replica of the file, and automatically cascading the edit to all of the replicas of the file.

In accordance with yet another embodiment of the invention an online file editing method is provided. The method includes uploading the file from a first client to a server, providing to the first client and a second client simultaneous access to a replica of the file, a first replica and a second replica, respectively, receiving from one of the first client and the second client an edit command to edit one of the first replica and the second replica, editing the one of the first replica and the second replica of the file according to the edit command, and automatically editing or causing to be edited an other of the first replica and the second replica of the file according to the editing of the one of

the first replica and the second replica of the file such that the editing is performed at a same location of the other of the first replica and the second replica of the file as it was performed in the one of the first replica and the second replica of the file.

In accordance with an embodiment of the invention a system to edit a file over a network is provided. The system includes a first client, which includes a first file and an editor that allows a user of the first client to access a replica of a second file, a second client that includes the second file and an editor that allows the second client to access a replica of the first file, and a server that provides the first client and the second client with simultaneous access to the replicas of the first and second files.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 depicts an exemplary computer network suitable for practicing the invention.

Fig. 2 depicts an exemplary flow diagram of the processing performed relative to this invention.

Fig. 2A depicts an exemplary screen shot of a form that can be used to create a session.

Fig. 2B depicts an exemplary screen shot of a page that can be used to upload a file from a participant's local drive to a server.

Figure 3 depicts an exemplary screen layout of a virtual meeting space tool interface in a two user implementation in accordance with the present invention.

Fig. 3A depicts an exemplary screen shot of the screen layout depicted in Fig. 3.

Fig. 3B depicts an exemplary screen shot of the screen displays of both participants of a two-participant session in accordance with the present invention.

Fig. 4 depicts an exemplary screen shot of a multi-user interface.

DETAILED DESCRIPTION OF THE INVENTION

This invention provides an interactive web-based communications tool which serves as an online, multi-directional file based streaming editor. This "virtual meeting space tool" allows multiple remote users to collaborate in real-time such that the users can compare, merge and instantly edit a file, while simultaneously communicating with each other online. The merging or edit performed is immediately and simultaneously "cascaded" to all session users. The term "cascade," as used herein, refers to automatically sending, or causing to be sent, an edit made in one file to all other files of a session, ensuring that the edits are made in the same place in each of the files. The process of cascading thus may include receiving an edit command, creating one or more packets reflecting the changes made to the file, compressing and encrypting the packets, sending the packets to all replicas of the file that are included in a session, ordering the packets at a destination, and editing the file according to the edit command, while ensuring that the edit is applied to an appropriate part of a file (i.e., the edit is made at a point of the file that corresponds to a point of the file where the edit was initially made). It is important to note that each session participant accesses a replica of a file, as distinguished from a copy of the file, which implies that an original exists and the others are copies of the original. According to this embodiment of the invention, each replica of

the file is continuously updated and maintained consistent with each of the other replicas of the file, as modifications to a replica of a file are automatically cascaded to all other replicas of the file.

An "initiating" user initiates and session, which can be joined by one or more "invitees." Collectively, the initiating user and invitee(s) are referred to as "participants." Any of the session participants can be in control of a session and therefore function as a host of the session. Each session creates a dedicated link among session participants. The initiating user can create and delete a session. Invitees may join scheduled or ongoing sessions. Each session is uniquely identifiable according to information that is maintained by the system, for example, in a database. Further details on creating and joining sessions are provided below.

In particular, the invention allows multiple users to simultaneously access a file, while one of the multiple users can edit the file at a time. The system explicitly indicates which of the users has editing control at a given time, and allows the user with editing control to transfer such editing control to another user as desired, i.e., the host user can change throughout a session. As a file is edited, changes are automatically and immediately cascaded to other session participants. Therefore, changes to a file can be viewed in real-time by other session participants. The virtual meeting space tool also provides additional editing features, such as, for example, tracking of changes, spell check, save, print, etc.

More specifically, the virtual meeting space tool provides a single interface that allows multiple network users to transfer files among one another, simultaneously open the files on each of the users' respective computers, edit the files, discuss the files via a

one to one discussion interface, and browse the Internet while performing the above functions. The virtual meeting space tool interface includes an editor, a "switch" utility which allows a single user to retain control of a file at a given time, a chat area, a reference area, and an instant messaging utility, each of which is described further below. A user can therefore open a file and, in a parallel frame, open a file that has been received over the network. The interface provides an editor which allows users to perform a variety of editing functions on a file.

The virtual meeting space tool in accordance with the invention provides various operational advantages that are not provided by conventional systems. For example, the virtual meeting space tool allows users to compare, merge and edit his or her own files, as well as the files of other session participants. Edits made to a file are transmitted simultaneously to all session participants, without a user initiating a "send" or "transmit" action. All session participants can save or print a file during the session. Further, during a session all participants may open, create or edit a file that cannot be viewed or otherwise accessed by other session participants. Such files can be merged with a shared file, regardless of the original owner of the shared file. The merged file is automatically transmitted to other session participants, as would a file that has been edited via the virtual meeting space interface. A "discussion area" section of the interface provides a chat function that allows participants of a session to discuss over files in real-time. The system further includes a manner for the users to designate one of the participants as having control over a file at a given time. The participant who has control can merge, edit, save, or print a file, and transfer control of the file in its full form to another session participant. Additionally, the invention allows an owner of a file to assign access

permissions to replicas of the file. For example, a file owner can provide other session participants with permission to edit the file, but deny such participants permission to print or save the file, or vice versa. Further, in an embodiment, the invention can automatically compare two files and indicate the differences on the display.

The files may be in a variety of file or media formats, including, for example, text, hypertext markup language (html), graphics, AVI, etc. In some embodiments, the tool may also enables users to browse the Internet while accessing shared files. In an embodiment of the invention, a user can open a file in one part of a screen and open in another part of the screen a file that was forwarded to the user during the current session, and work on both simultaneously. Similarly, in an embodiment of the invention, session participants can copy text and objects from each other's files and paste them in their own respective files, thereby editing their own files based on the simultaneous discussions and suggestions taking place through the discussion area without affecting the shared files. After editing, both users can save, print, or download the file and the discussion that occurred in the discussion area.

Figure 1 depicts an exemplary computer network 100 suitable for practicing the invention. Network 100 includes clients 105 and 110, which communicate with one another and server 115 via network 120. Clients 105 and 110 preferably correspond to desktop computers. Network 120 corresponds to any public or private network, such as, for example, the Internet or a LAN. Each of clients 105 and 110 include the conventional components of a computer including memory, at least one processor, storage, and input and output devices. One of skill in the art will appreciate that while clients 105 and 110 have been depicted in Fig. 1 with a storage device, the invention may be practiced with

dummy terminals that do not include storage devices. Specifically, in such embodiments, no stored file is necessary. A document can be prepared, online or offline, or a user can join a session without contributing a file. The memory further includes one or more applications 118 that are used to create and open files that are transmitted and displayed according to the interface of the invention and a virtual meeting space tool interface 125 that allows files to be opened, edited, and shared with session participants. The applications may include, for example, text, graphics, or spreadsheet applications. Session participants can upload locally stored files to server 115, which are downloaded to each session participant's computer when the file originator opens the file via the interface 125. An edit function of the virtual meeting space tool interface 125 provides a single user with permission to edit a file. All other session participants are only provided read access to the open session files. Each session participant has access to a local replica 138 of other session participants' files. The interface 125 will transfer edit control to another participant upon receiving an appropriate input. The virtual meeting space interface 125 is user friendly and includes, for example, a series of pull down menus, forms and selectable items.

The editor supports a variety of edit functions including, for example, typing, formatting (e.g., bold, italic) and clipboard operations (e.g., cut, replica, paste), and operations that respond to keyboard and mouse events, operations performed via a toolbar, and operations performed via a mouse click. The editor determines where an edit occurred and also determines the properties of the edit, e.g., color, font, format, etc. It also supports opening and saving of files that include both formatted and non-formatted text. The editor may further include a speech recognition feature. The editor opens text

files in a rich text file format (RTF) such that the location of each character of a file is uniquely identifiable. Thus, whenever an edit command is received, the system performs the edit on the replica of the file on which the edit is made and then makes the same change in the same location on all corresponding replicas. Similarly, in a spreadsheet implementation, each cell of a file is marked according to a unique identifier corresponding to a cell address and for a graphic file, lines of marking uniquely identify screen coordinates of the file. The cascading process receives an edit from a marked place of a file and sends the edit to the same marked place of a replica of the file.

Because the editor works according to type and location of an edit, it is language independent. Therefore, the editor can handle fonts of any language that a user has installed on his or her computer. The editor also allows each of the session participants to edit local files, i.e., files other than the shared replicas, during a session. For example, if one participant has edit control, while that participant is editing a replica of a file, the other participant may be working locally on a file that is stored on the hard drive.

Server 115 also includes the conventional components of a computer including memory, at least one processor, storage and input and output devices. The memory of server 115 further includes virtual meeting space tool 130 which allows multiple users to access files simultaneously. The tool 130 coordinates interaction among users via the interface 125. Files that may be simultaneously accessed by the multiple users are uploaded from a user's local storage 140 on a client 105 or 110 to server 115 for virtual storage as file 150. The virtual meeting space utility 130 is not language or application dependent. It is operable with any application included on clients 105 and 110.

Figure 2 depicts an exemplary flow diagram of the operation of a virtual meeting space tool in accordance with the present invention. A initiating user begins by creating, i.e., scheduling, a session (210), i.e., the user who creates a session is also the initiating user of the session. Fig. 2A depicts an exemplary screen shot of a from that can be used to create a session. A session is identified according to a unique identifier and has a begin time and an end time. A session may therefore be represented, for example, by an application level array that includes a title, a description, one or more invitees, an initiating user, and a session application number. When creating a session, a system user provides the necessary information via the system interface. The creator of a session indicates each of the desired session participants because only indicated participants will later be allowed to join the session. The creator of a session can amend the list of invitees to reflect additional participants at any time between creation and deletion of the session. After a session is created, the system notifies each of the invitees of the session, for example, by sending an instant message to each of the invitees, informing them of information about the session, including its name, initiating user and time of operation.

In this embodiment, to begin a session a session participant uploads one or more files to a server (220). Fig. 2B depicts an exemplary screen shot of a page that can be used to upload a file from a participant's local drive to a server. The virtual meeting space tool interface on each user's computer performs this upload function. The files that are uploaded to the interface can be accessed by any of the participants of a particular session. Once a file has been uploaded, it is treated as a virtual file in that a replica of the file resides on the server until it is opened by its originator. When a file is opened, a replica of the file is downloaded to a memory area of each of the session participants'

machines. It should be noted that the session participants may upload additional files to the server as desired throughout the session. Additionally, session participants may create new files or download files retrieved from an Internet browsing session and upload the files to the server. All such files may be viewed by other session participants when the originator opens the files.

Once an uploaded file (which resides at the server in this embodiment) has been opened, a replica of the file is downloaded to the local interface of each of the session participants. As described above, each of the session participants has full access to shared files, including editing, saving, merging and printing capabilities (230). As described above, during a session, changes made by each participant are automatically cascaded to other session participants. A single session participant has editing control of all session files at a given time during the session. The session participant in control can save, edit, merge, or print a file. When a session time lapses (240 and 270), the session may be closed or extended. Prior to closing a session, the system provides each participant an opportunity to save the replica of the file.

Additional participants may join a session at any time (250 and 260). A user may joins a session by providing an appropriate input to the system, for example, by selecting an icon on the system interface. Only users who are listed as invitees to a session may join a session. Therefore, if a user attempts to join a session and the user is not indicated as an invitee to the session, the system denies the user access to the session. Each time a new user joins a session, the system automatically reformats the display of all participants to reflect the newly joined participant. A user may join a session at any time during the duration of the session.

After the session time has lapsed, as indicated above, the session automatically terminates (270). While a session can be deleted by its creator at any time after the session has been created, this exemplary flow diagram illustrates a session deletion as occurring after the session has been terminated (280). To delete a session (290), the session initiating user selects an indicated session from a list of sessions and deletes it. When a session is deleted, all files associated with the session that are stored on the server are also deleted. Receiving and processing the information relative to creating, joining and deleting sessions is well-known in the art and therefore not described in further detail herein.

Figure 3 depicts an exemplary screen layout of a virtual meeting space tool interface in a two user implementation in accordance with the present invention. Fig. 3A depicts an exemplary screen shot of the screen layout depicted in Fig. 3. As depicted, the interface simultaneously displays both the current user's file and another user's files, 310 and 320, respectively. Each of the session participants can edit any of the open files in the session according to the editing functions provided on edit tool bar 325. Each session participant can open a single file at a time, although each session participant can open multiple files during a session. Files can be edited by the participant that has edit control at a particular point in time. Control indicators 327(a) and (b) correspond, for example, to selectable buttons that both indicate which participant has control and provides that control to the indicated participant. The participant having control is the only session participant who possesses complete file editing capabilities. Thus, control indicators 327(a) and (b) indicate whether a particular participant can perform edits or not. For example, the control indicators 327(a) and (b) may be of different colors such

that a green indicator 327(a) indicates that the user 310 can perform editing operations on all session files and a red indicator 327(b) indicates that the user 320 only has read access rights to all of the open session files. The edit functions of all participants not having control are effectively locked such that the participants have read only access to the open files. Read only file access includes scrolling operations.

Edit tool bar 325 includes basic editing operations including, for example, clipboard operations such as cut, paste and copy, drawing functions, formatting, etc. The editor also includes a function that records editing operations performed by the user. This function therefore supports undo, redo, and track change editing operations. Each time an edit is performed, the edit data is cascaded over the network and transmitted to each of the files included on a computer that is participating in the session. To appropriately cascade data edits, the system determines the following information: (1) the edit; (2) the type of the edit, e.g., type formatting, a clipboard operation, etc; and (3) where in the file the edit occurred. The virtual meeting space utility compresses and encrypts the cascaded data to ensure that it is sent according to an optimal transmission rate. Conventional compression and encryption techniques are used to compress and encrypt the data.

When transmitting data across the network among computers, the system ensures that the data is consistent and that lost data is tracked and restored. The system uses a sequential data checking mechanism in which each data packet sent from one node to another is sequentially numbered. Any discrepancy in the order of received packets indicates a data loss. When a data loss is detected, a data restoration mechanism is used to retrieve the lost data from the originating machine. In this embodiment, when a data

loss is detected, the control indicators 327(a) and (b) are turned to yellow to indicate such data loss. During this time, none of the session participants can edit a file. When the data loss has been corrected, i.e., when the data restoration is complete, the control indicators 327(a) and (b) are returned to their previous color and edit control is returned to the previously indicated participant. One of skill in the art will appreciate that there are different means of indicating data loss, such as, for example, a speech prompt.

Screen area 330 provides an area where a user can list all of the files that have been uploaded to the server. Each session participant can upload multiple files to the server. Once a file is opened, it is downloaded from the server to the local machine of each of the session participants. Edits to the file are thus made on a local replica of the file. As edits are made to a replica of the file, the edits are automatically cascaded to all other replicas of the file. Each time a new file is opened, the current file is closed. Prior to closing a current file, the system allows the participant in control to save the file. The participant in control can provide such control to other participants so that they may save the file as well. Screen area 340 corresponds to a discussion area, where participants can engage in a real-time chat session, transmitting text among each other. For example, the screen layout could conform to a side-by-side arrangement of 310 and 320, and 330 and 340, respectively.

Fig. 3B depicts an exemplary screen shot of the screen displays of both participants of a two-participant session. In the example, the display screens of the two participants are swapped such that screen areas 310 and 320 are reversed on the screens of the users, i.e., the top portion of each user's display screen includes the user's file. Whichever participant has edit control, however, can edit both files included in screen

areas 310 and 320 via edit toolbar 325. Each participant has a different list 330 of files that were uploaded to the server. And each user's display includes control indicators 327, a discussion area 340, and an instant messaging area 350.

Fig. 4 depicts an exemplary screen shot of a multi-user interface. A multi-user interface may be: (1) the same format of the two user interface described above, where user may open a file, which can be accessed by all other participants. In this implementation, the interface would include the same screen components, although each of the screen areas would be of a smaller size; or (2) a single file that has been made accessible to multiple session participants. In this implementation, as described above, a single participant has edit control at a given point in time and such control can be easily transferred to another participant. Additionally, each participant has a replica of the file, to which any edits made to another replica of the file are automatically cascaded.

In the exemplary interface depicted in Fig. 4, a single file is displayed. The screen of each participant in the session depicted in Fig. 4 would closely resemble the exemplary display that is shown. The file is displayed on the left hand side of the interface. An edit toolbar 415 is included at the top of the file display area. Multiple invitees, listed at 420, can access the file. For each invitee, the system indicates the following: status, switch, and selection. The status area indicates whether the invitee is participating in the session. The switch area indicates which user has edit control at a given time. The participant having edit control can transfer such control to another session participant by selecting, for example, interrupt 425. The remove area allows a session participant to indicate to other participants that he or she is leaving the session. 430 and 440 correspond to additional areas where a session participant can browse, open

and send additional files, and chat with other session participants. One of skill in the art will appreciate that these screen shots are for illustrative purposes, and that an actual screen may include additional or different features.

Although this invention has been described relative to a particular embodiment, one of skill in the art will appreciate that this description is merely exemplary and the system and method of this invention may include additional or different components. This description is therefore limited only by the appended claims and the full scope of their equivalents.

Further details of the virtual meeting space tool are provided below. The following description further explains implementation details of novel aspects of the invention. In the following description, the term "buddy" refers to an invitee, "Switch Notes" refers to the invention, and "switch" refers to transferring control among session participants.

Switch Notes – A Combination – The Technical Implementation.

Switch Notes is a unique concept and product and a new and novel application which allows users at different locations in the world to edit any document over the internet as well as any intranet system. The edits made by one user on a document are caused to happen in all the other replicas of documents opened in a session at different locations almost instantaneously by the Switch Notes application. The system is unique both at usage interface level as well as the technical level. At the usage level, the uniqueness comes from the presence of utilities like cascading editor, its editing, comparing and merging abilities over the internet, the Switch, chat area, the Reference area, and the Instant Messenger – all present in the same, simple interface.

At an internal level, it is also an effective combination of various technical modules some of which are novel. The uniqueness in this product from the point of view of developing the product lies in the fact that a series of issues which have to be solved to make it happen. Following are the important technical modules that one will have to incorporate in order to develop such an application.

1. **Editor**. The first and the most basic need is to develop an editor that will provide with basic editing operations. Switch Notes has a root editor that provides the basic editing operations to a user on a stand alone basis.

2. **Recording an Edit**. The next important phase is to build the capability to record the editing operations done by the user. This had to be done in a way that all possible editing operations are recorded within the switch notes software.

3. **Cascading of Edits.** This is the most crucial part where the edits are to be cascaded. Under this section, the following points are important:

    a. To be able to cascade data over the Internet or intranet to any computer that has an access to the Internet or intranet as the case may be.
    b. To make sure that the data cascading happens with the most optimized time rate. For this appropriate level of compression was needed to be to done.
    c. To make sure that the data that is being cascaded is secure through the transfer channel. For this appropriate level of encryption mechanism need to be adopted.
    d. To make sure that the data is cascaded at the right place as in the master document. For this it was to be identified that (1). What is the edit? (2). What is the Edit Type? And (3). Where is the edit? By edit type, it is meant that whether it is a typing, or formatting or a clipboard operation (cut/copy/paste). Gathering all this information, it became easy to cascade the edit to the destination editor.

4. **Data Consistency.** It was very important to make sure that the data traveling between nodes is consistent and that no loss is happening on the way. For this a sequential data checking mechanism was needed. Each data packet sent from one node to another was numbered. Any discrepancy in the order of received numbers

indicates a data loss, in such a case, a restoration mechanism tries to retrieve the data again from the originating end.

5. **The Switch.** To make sure that the Switch transfer happens for the appropriate destination editor of the buddy online. Also, to make sure that a green/red combination (i.e. edit/non edit) is always maintained during usage.

6. **Locking of Editor.** To make sure that the locked editor does not accept any editing operations from the user. But, at the same time, to allow the readability and scrolling features on this editor.

7. **The Browse Area.** Switch Notes allows for opening for a reference document that will not be shown to the buddy at all. In the Browse Area, the use can open any document from own computer and use it as a reference document for the online discussion ongoing on the Switch Notes editors. The user can do comparison / merging of the documents open in the Switch Notes editor and with this stand-alone reference document.

It is the combination of all the above-mentioned points that make Switch Notes one of its kind applications. All the solutions have been placed in the system in a well-designed fashion.

1. Editor

Building of an editor that tells the system what has happened was the first part of the development. The following were the key requirements of the editor:

(1). It should allow typing,
(2). It should allow formatting operations (like bold/italic etc),
(3). It should allow clipboard operations (like cut/copy/paste etc),
(4). It should allow operations that respond to keyboard and mouse events,
(5). It should allow operations through a toolbar,
(6). It should allow operations through a right click menu of mouse.
(7). It should tell the system where the edit happened,
(8). It should tell the system what are the properties of the new edit happened (i.e. color/font/format etc),
(9). It should allow opening and saving of documents that contain formatted as well as non-formatted text.
(10). It should allow speech recognition.

Following is a detail of how each one of the above was achieved.

(1). Being an editor control, it automatically allows typing.
(2). Such operations can either be invoked through toolbar/keyboard/right click. The key thing was to identify the property that would make this happen. For this purpose, one made use of properties like SelBold, SelItalic, SelUnderline, etc available in the EditControl. The behavior of these properties would vary with changing their value from FALSE (default) to TRUE.
(3). Clipboard operations through the keyboard (ctrl C, ctrl X, ctrl V) are by default supported by this control. But to handle such actions through toolbar and right click menu, one made use of the Clipboard object. The system used the methods SetText and GetText of Clipboard object to perform the clipboard operations as needed.
(4). Being a basic editor control, it does respond to keyboard and mouse events.
(5). For this, one had to make an imagelist control with a number of pictures acting as buttons in it that would further correspond to the desired action.
(6). For this, the right click of the mouse was trapped in the mouseDown event, and then invoking a menu item that further contains the desired sub menus for the required actions.
(7). Editcontrol has a property called SelStart that tells the system where the edit has happened. The basic assumption here is that the numbering of characters of any document will never vary. In other words, if one has a document that contains a word "switch" at number 10 in the document, then it will remain at the same number till the document is changed.
(8). The properties of the new edit are exposed by SelFontName, SelFontSize, SelColor, SelBullet etc. All these properties (called SetA), constitute to tell the system about what the new edit looked like.
(9). The EditControl has methods LoadFile and SaveFile that allow opening of a RTF (Rich Text Format) document into the editor. As a pre-process to the Switch Notes Session, while uploading the document to Server Archive, the document is converted to RTF by instantiating a Word.Application object. This conversion also makes sure

that Switch Notes editor can then allow online editing online of various types of documents.

2. Recording of Edits.

Edit recording was done in various events of the editor, namely the Keypress, keydown, keyup, mousedown, mouseup events. The character set of the keyboard is covered through the KeyPress event. The non-character set of the KeyBoard is covered by the KeyUp and KeyDown events.

Basically, and edit can be one the following:

(1). Typing
(2). Formatting
(3). Clipboard operations

Event handlers written as mentioned above gather all the SetA properties, the positional properties (selStart, selLength), and the data of the edit, merge them into a large String of information. This String is known as the Command String. The Command String is the key to the whole system. This carries all the information needed for the receiving editor to carry out the corresponding cascading operations.

FLOW CHART No. 1 explains this phenomenon. The Flow chart connectors ET (Editor Typing), EF (Editor Formatting), and ECP (Editor Clipboard) handle the preparation of command strings based on user's actions.

3. Cascading of Edits

The next thing was to dispatch the Command String to the destination(s) editor.

The Switch Notes control is embedded in a container (browser). There is a hidden Java Applet in this container. The ActiveX control and this Java Applet can communicate with each other using JavaScript.

The Command Strings travel through the ActiveX → JavaScript → Applet channel to the Switch Notes Server where a Java Server program processes these strings. Internally, each Switch Notes Session has a unique ID. Within each Switch ID, there can be user(s). The Java Server identifies the ID of the buddy to whom the data should go from ID of the originating user. If the buddy is connected to the server at this stage, the data is sent to the destination. The Java Socket programming is used in this data transmission. The flow chart connector EC (Editor Communication) in FLOW CHART NO. 6 describes how Java programs handle the transfer of data.

The Command String to be dispatched to this communication mechanism occasionally will contain the data in large proportions also like some pasted data or some document data. Passing such big amount of data over the Internet in its base form is never a good idea. Hence, to speed up the process, the data was compressed.

A high degree of compression was used for such Command Strings. At the receiving end, again the data is decompressed and presented to the destination editor in base form. Such data was also required to be encrypted for only Switch Notes to understand it.

At the receiving end, first the Command String is interpreted as to what the string is for. After this, the appropriate execution scheme is triggered. In general, any execution scheme will do the following:

1. Set the positional parameters of destination editor the same as origin (i.e. set the SelStart and SelLength)
2. Set the SetA properties
3. Print the data/Make the corresponding change

The FLOW CHART NO. 3 explains this execution procedure. The connectors EXED (Execution for an Edit), EXWD (Execution for Whole Data), and EXSW (Execution for a Switch) handle the execution of the incoming Command String.

A great feature of Switch Notes is that it can handle any language fonts. If, for example the users have a font of Russian installed on their computers, then they can do editing online in Russian language.

Both the users can continue to edit their own documents independently of the buddy's edits.

A similar cascading logic can be implemented for applications like spreadsheets, graphics. The key factor is to identify what is same in both the documents. In a text based document, the character numberings remain the same in both documents, in a spreadsheet, the cell address are the same, and in a graphic software, the screen coordinates are the same.

4. Data Consistency

Another problem that is encountered in data transmission process is to make sure that the data packets are sent and received in order of their generation. For this, a a numbering mechanism of the data packets has been prepared. Each packet (Command String) originating from one end to another was sequentially numbered. At the receiving end, the sequence of packets was noted on each receive operation. The instant any discrepancy in numbering is found, it is taken as an error, and a restorative mechanism triggers on to get the missed data back from the originating editor. In such a case, the Switch lights, are turned to Yellow to indicate a data loss. At this instant, no editing operations are allowed. On the completion of the restorative mechanism, the lights are brought back to the original stage. The flow chart connector DL-NUM in FLOW CHART NO. 5 explains this numbering scheme.

A concept of PRESTR and POSTSTR is also implanted. While preparing the Command String, the first5 and next5 characters at the point of edit are picked up and

termed as PRESTR and POSTSTR. They are merged with the Command String. The merging of POSTSTR AND PRESTR us explained in FLOW CHART NO. 7. At the receiving end, before executing the command string, the PRESTR and POSTSTR received in the Command String are validated against the ones already present around the point of edit. If these two strings are consistent, then it is ensured the edit will happen at the correct place. If any error is noticed in this phenomenon, then the light is turned to YELLOW and The Restorative phenomenon is again launched to bring status to normal. The flow charts DATA CHECK explains this checking mechanism (in the FLOW CHART NO. 8).

5. The Switch

The Switch is the indicator of control. Normally there is always a diagonal relationship of red/green states of the editor. By default, the top editors of each interface are GREEN, and the bottom editors of each editor are RED. In order to make sure that no editing is accepted by the editor, the LOCKED property of the EditControl is set to TRUE. In accordance with RED light, the top editor's toolbar is also made Locked.

Switch transfer is also treated as a Command String. This command string does not carry any user data. It only carries a flag saying TRUE. On receiving of this flag, it is interpreted that a Switch Transfer has happened.

The Switch Concept can very well be applied to any other application also whereby only one person can take control of the application at one time. Switch transfer would make sure that the Control is shifted to the concerned destination.

The three lights provided with the Switch act as a visual signal for allowing effective communications. While the green and red lights indicate the user's rights to edit the document, the yellow light works as a status indicator.

If there is a data loss encountered by the system, then the light turns YELLOW and remains yellow till the restoration happens. At this instant, no editing operations are allowed.

If the buddy goes offline, then the lights turn GREEN and YELLOW together indicating that the buddy has gone offline, but the user can still do the edits.

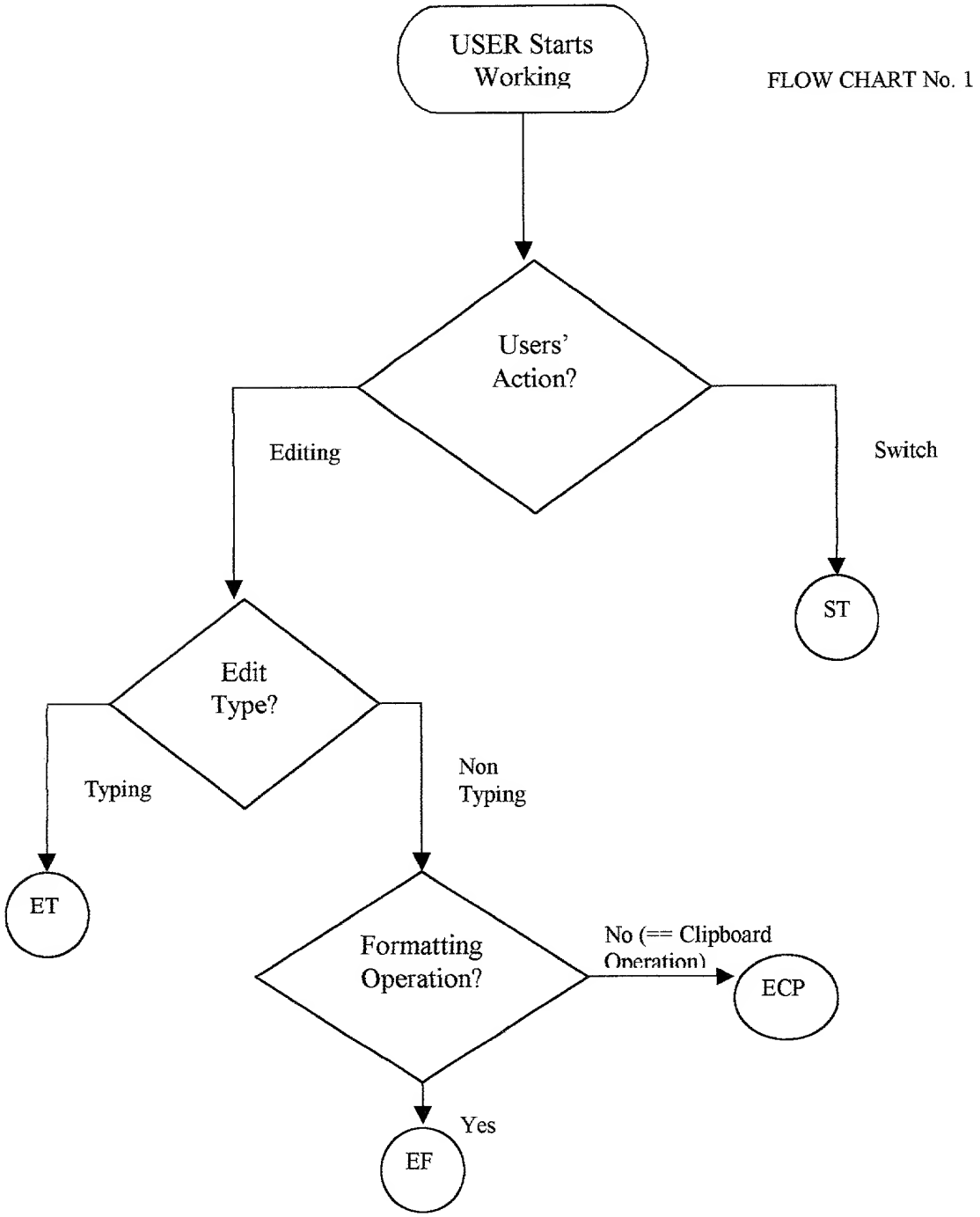
IF the light is RED or YELLOW alone, then editing is not allowed, but if the light is GREEN OR GREEN+YELLOW, then the editing is allowed.

The Flow Connector ST (Switch Transfer) in the FLOW CHART No. 4 explains the switch transfer phenomenon.

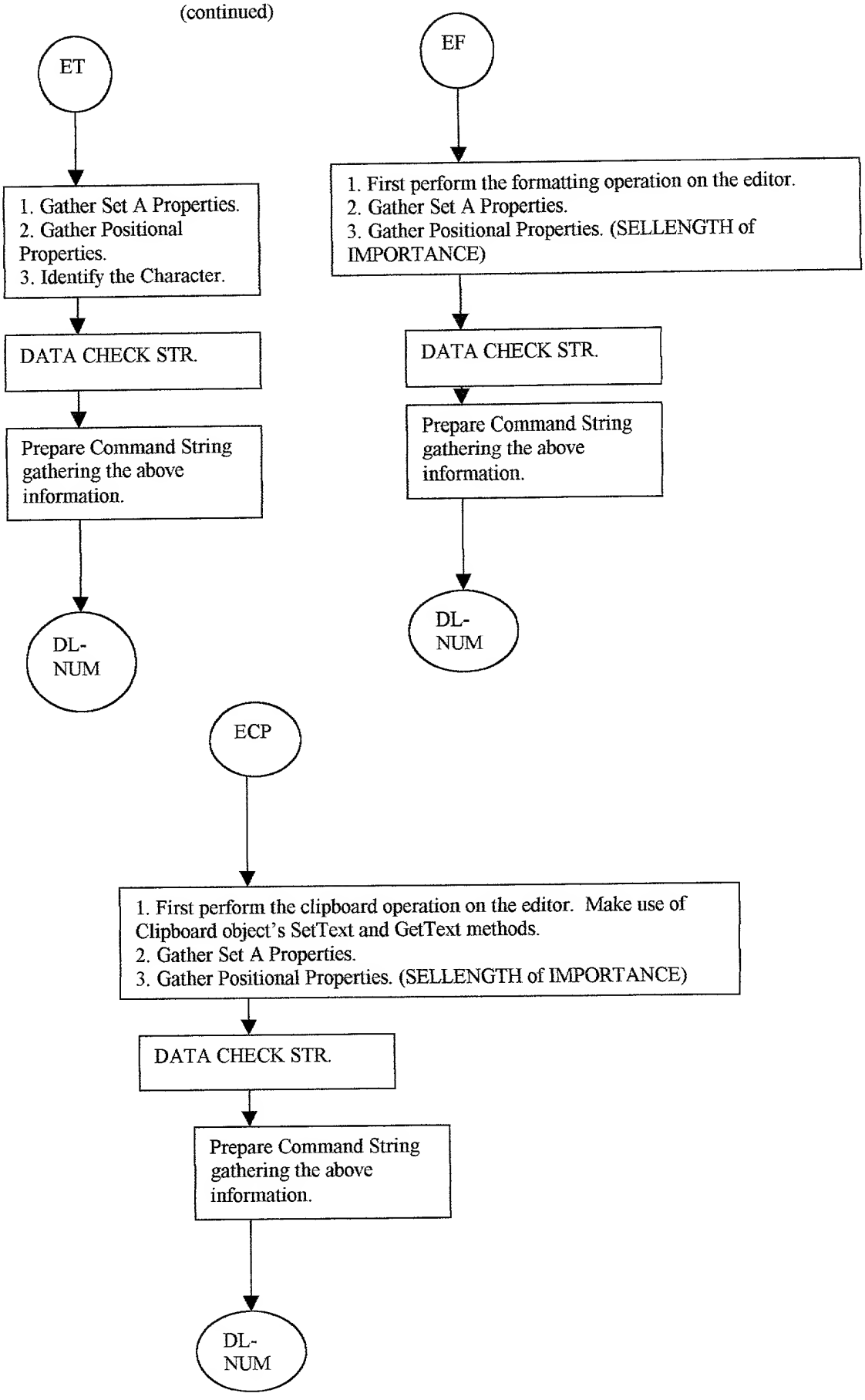
6. Locking of Editor

For this the Locked property of the EditControl is set to TRUE. This ensures that no editing is accepted by the editor.

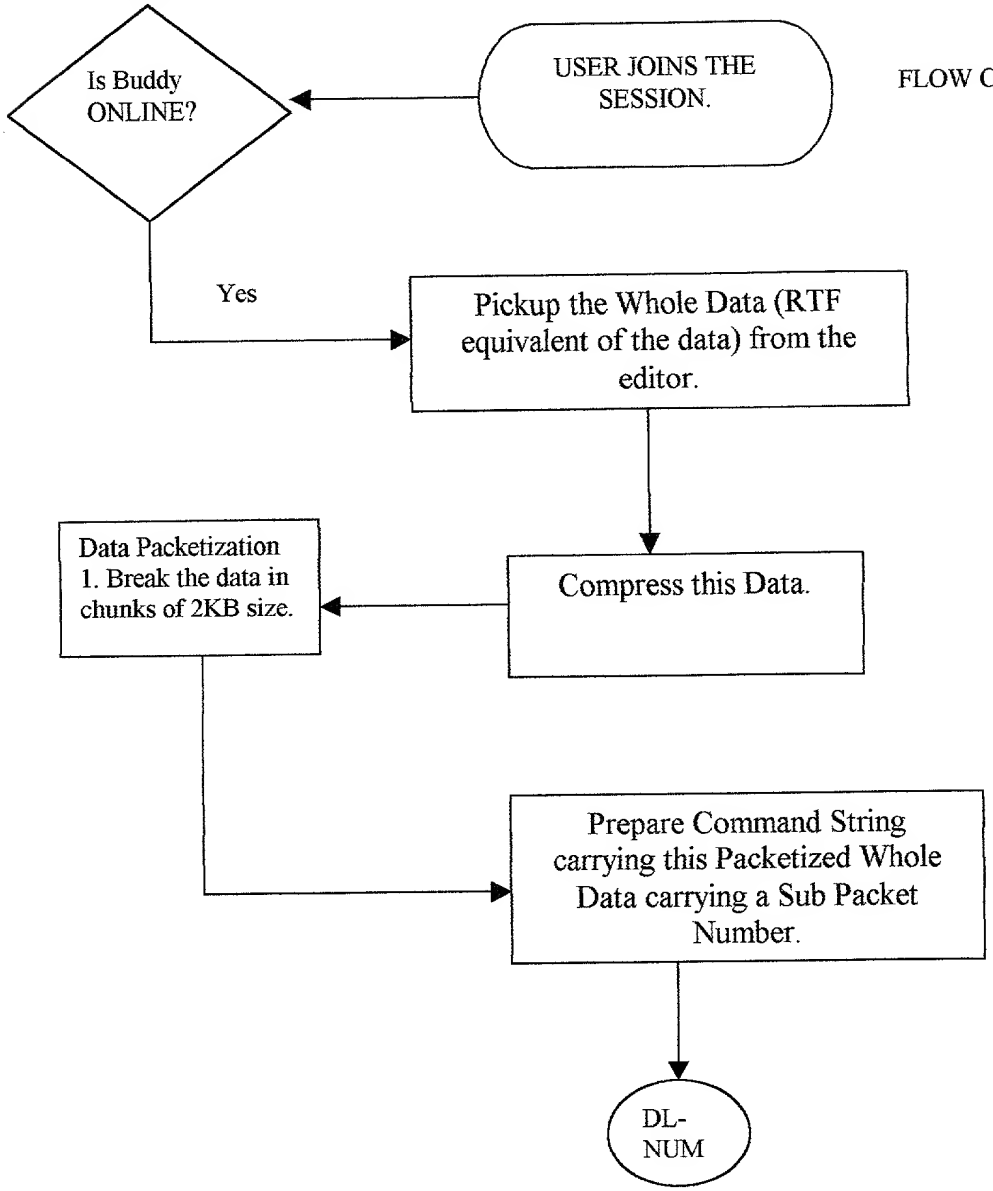
Preparation of Command String (User' Actions)



FLOW CHART No. 1
(continued)



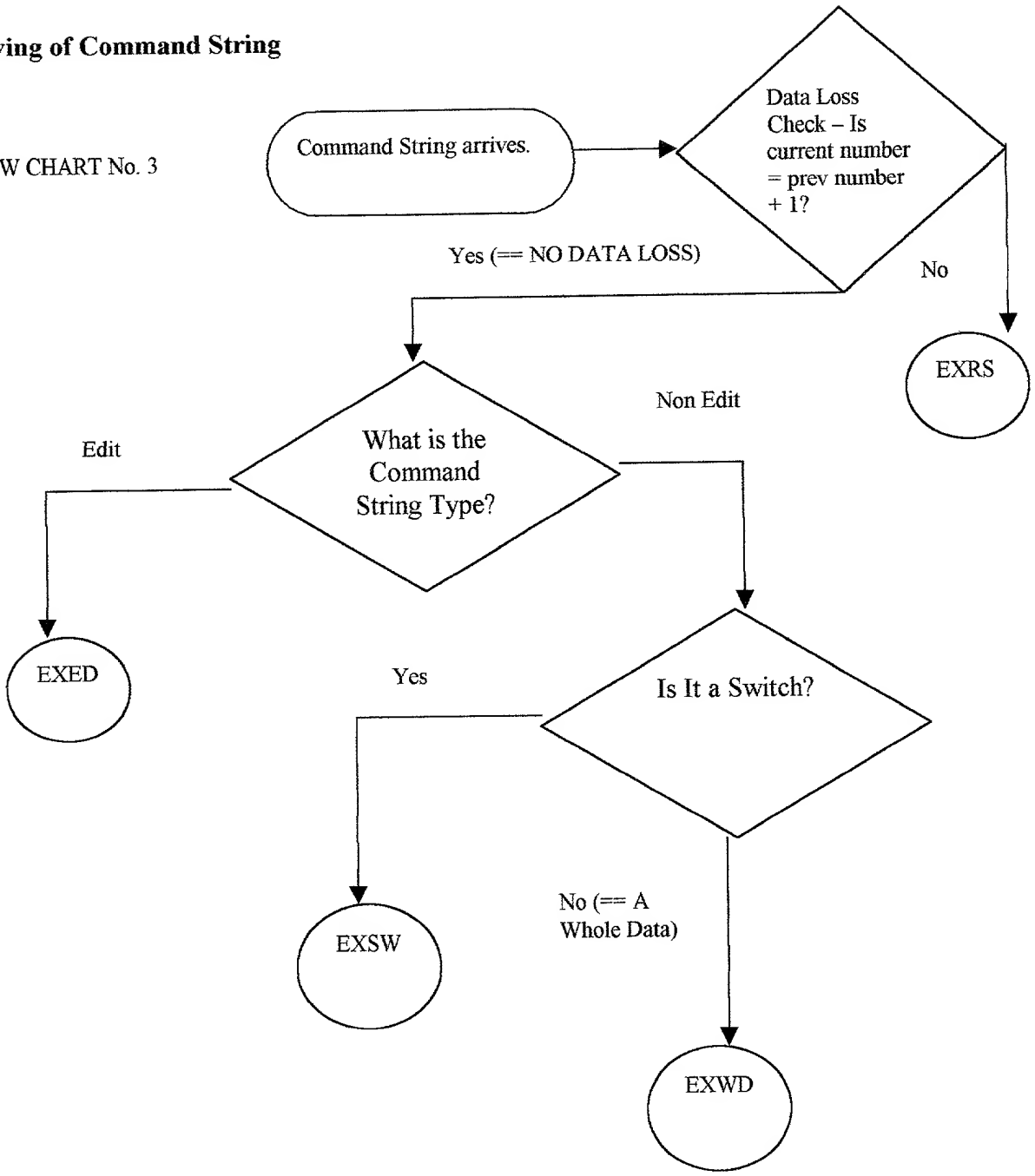
Initial Processes (When User Joins the Session)



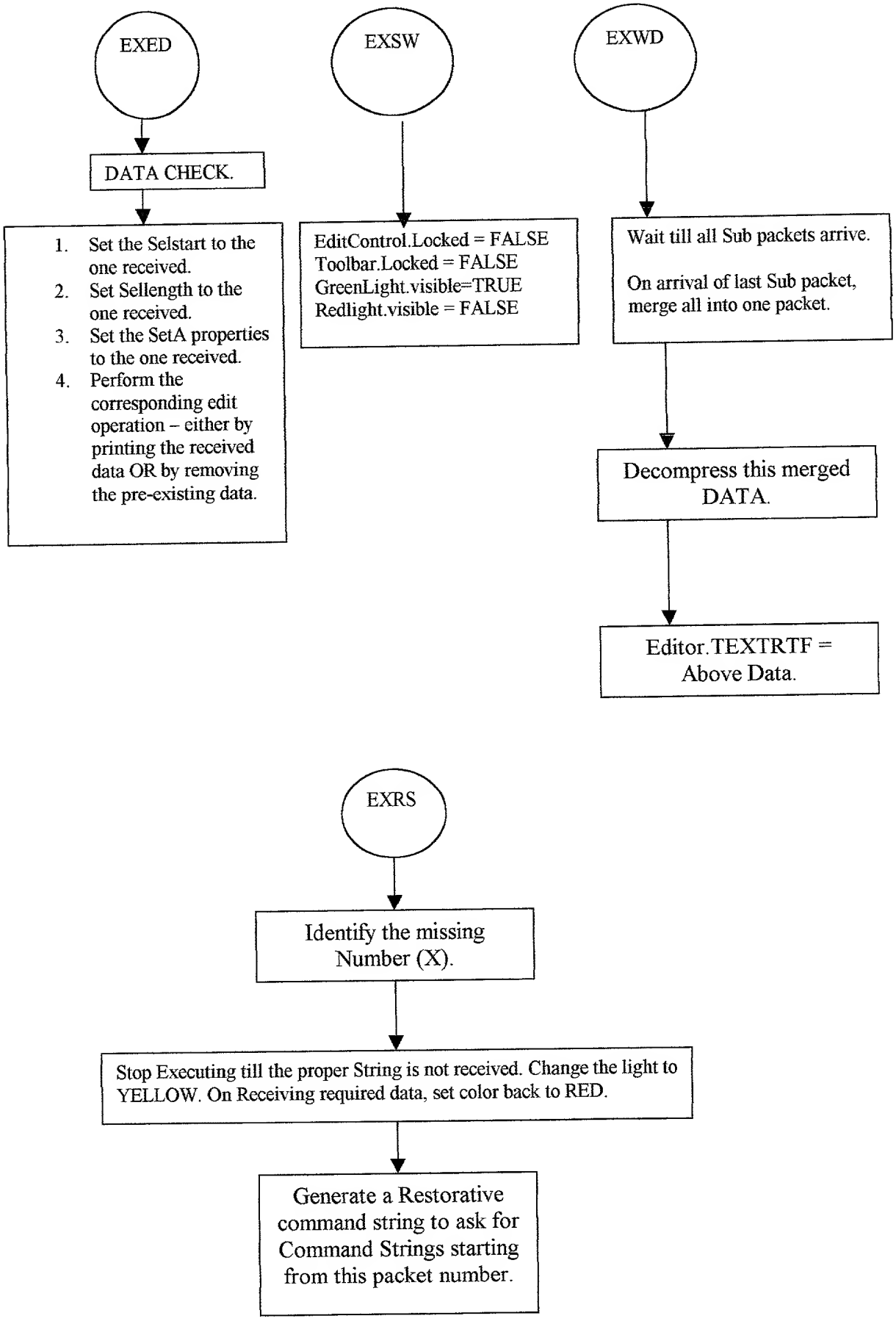
FLOW CHART No. 2

Receiving of Command String

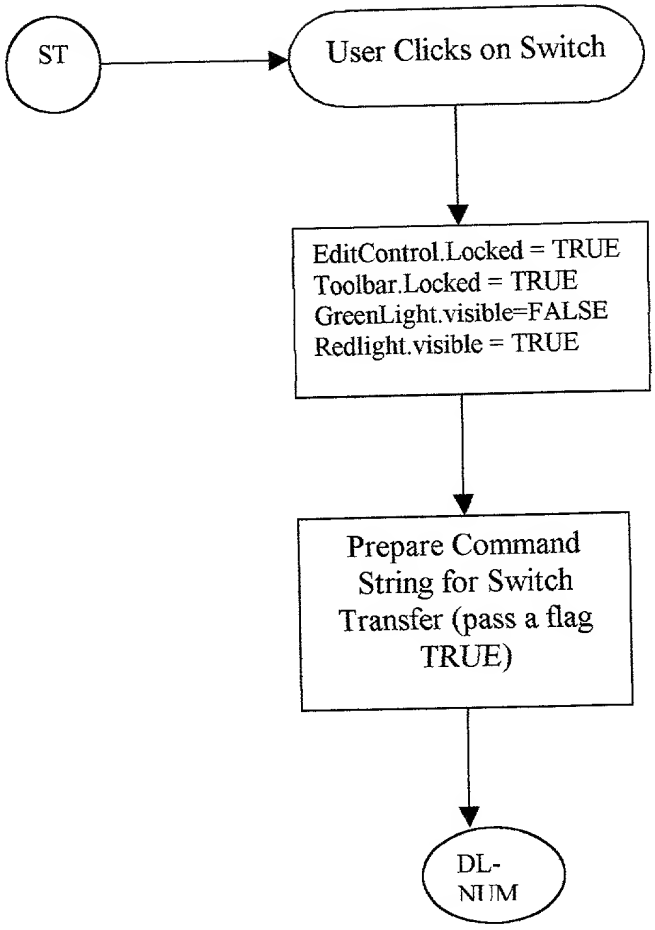
FLOW CHART No. 3



FLOW CHART No. 3
(continued)



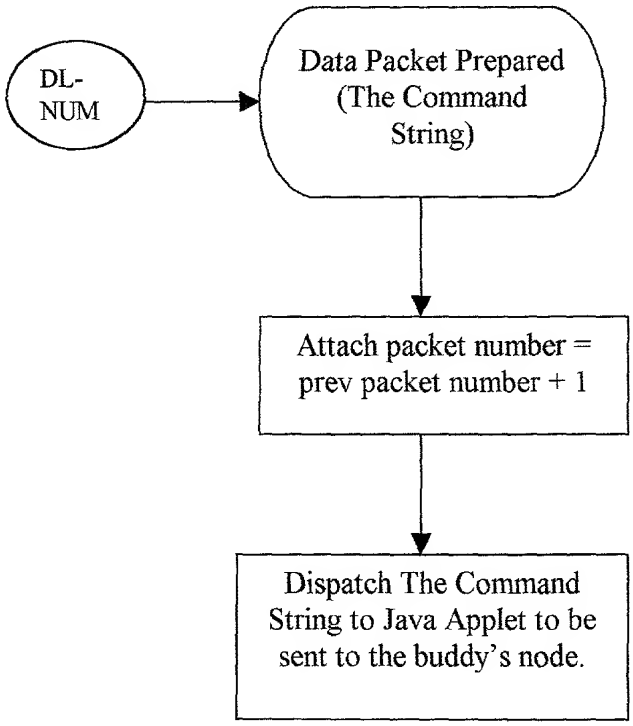
Switch Transfer



FLOW CHART No. 4

Data Loss (during transmission – adding numbering)

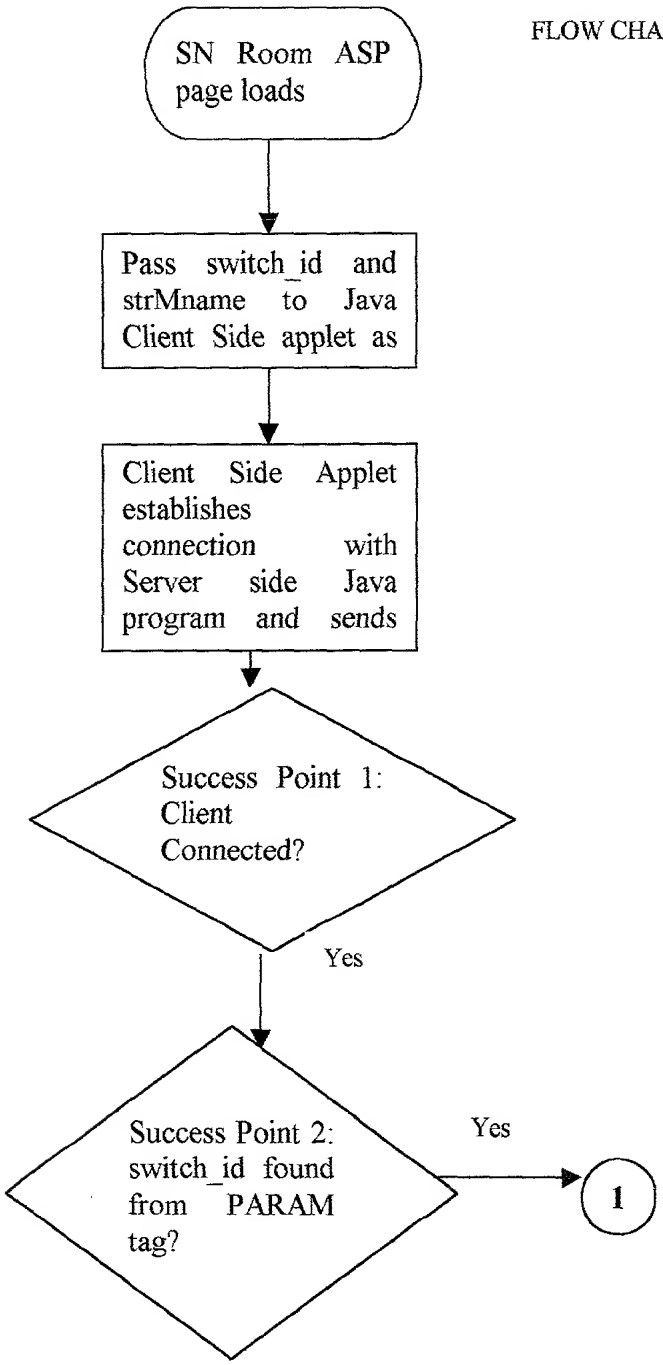
FLOW CHART No. 5



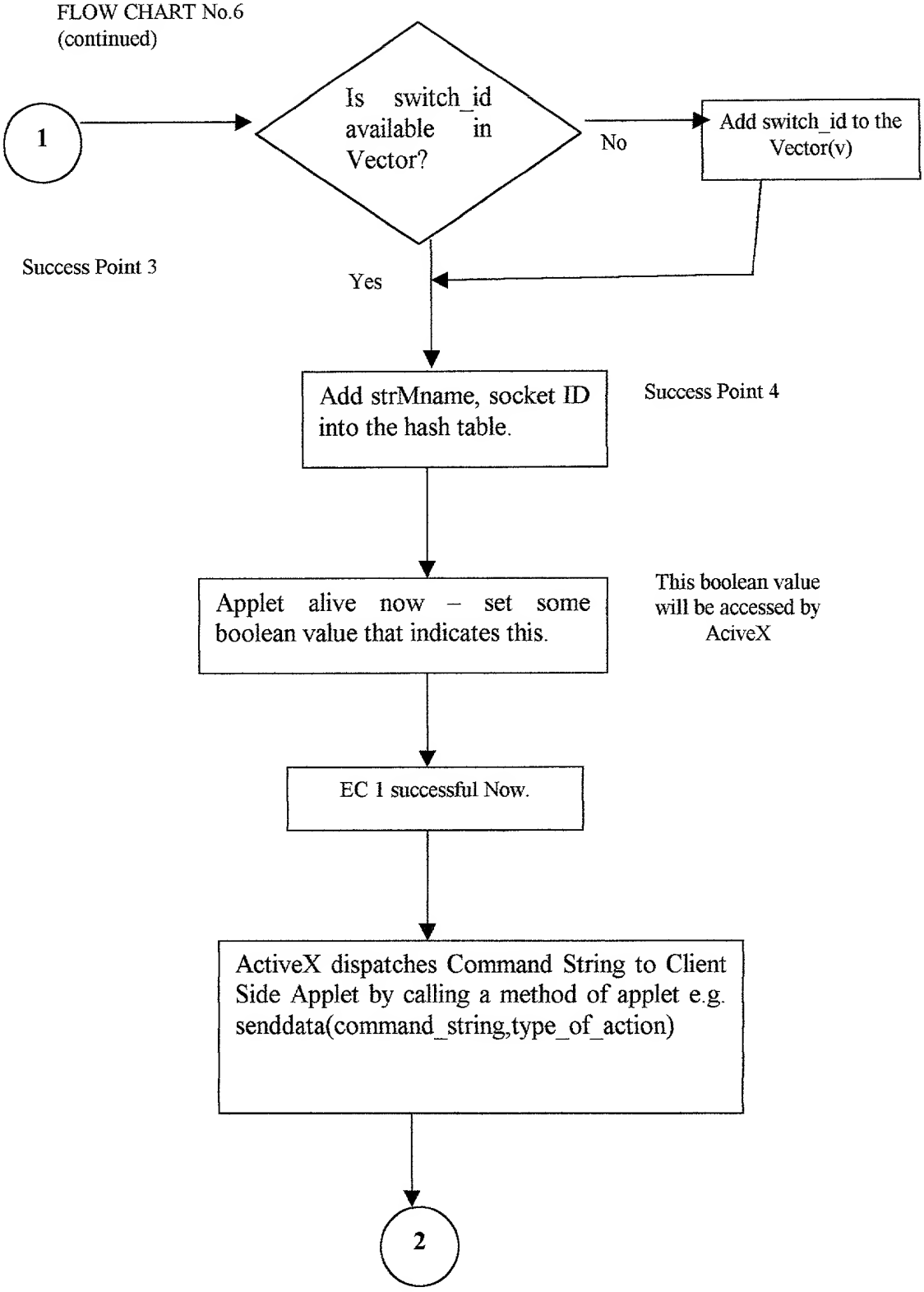
Sending of Data – Java Processing (EC – Editor Communication)

Following is a Flow Diagram for the EC related activities.

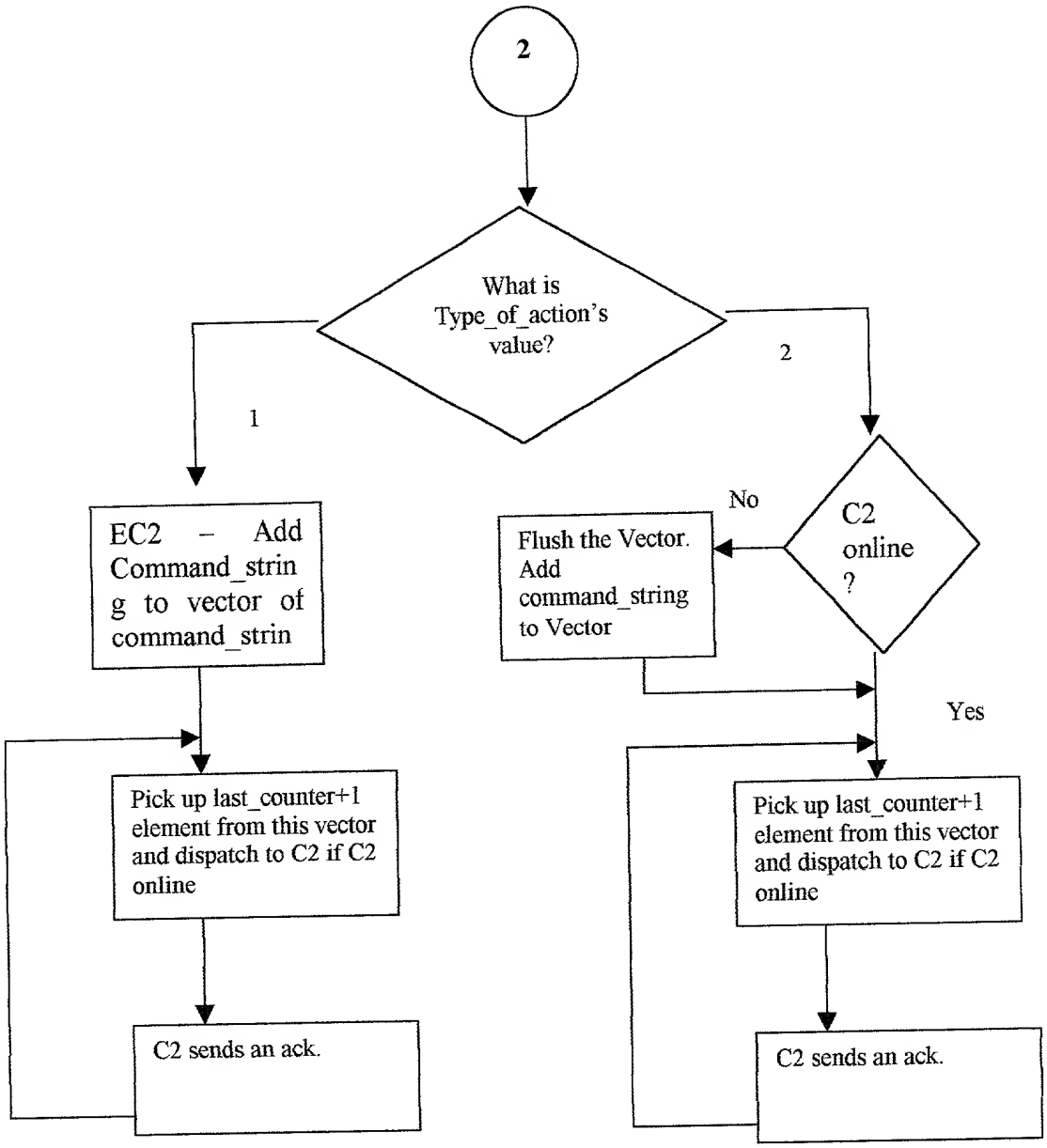
FLOW CHART No. 6



FLOW CHART No.6
(continued)



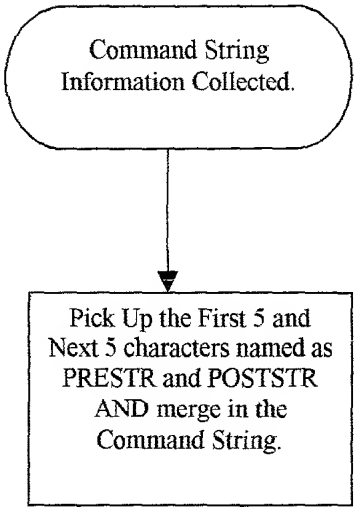
FLOW CHART No.6
(continued)



Data Loss (To check if the Edits are happening at the right place)

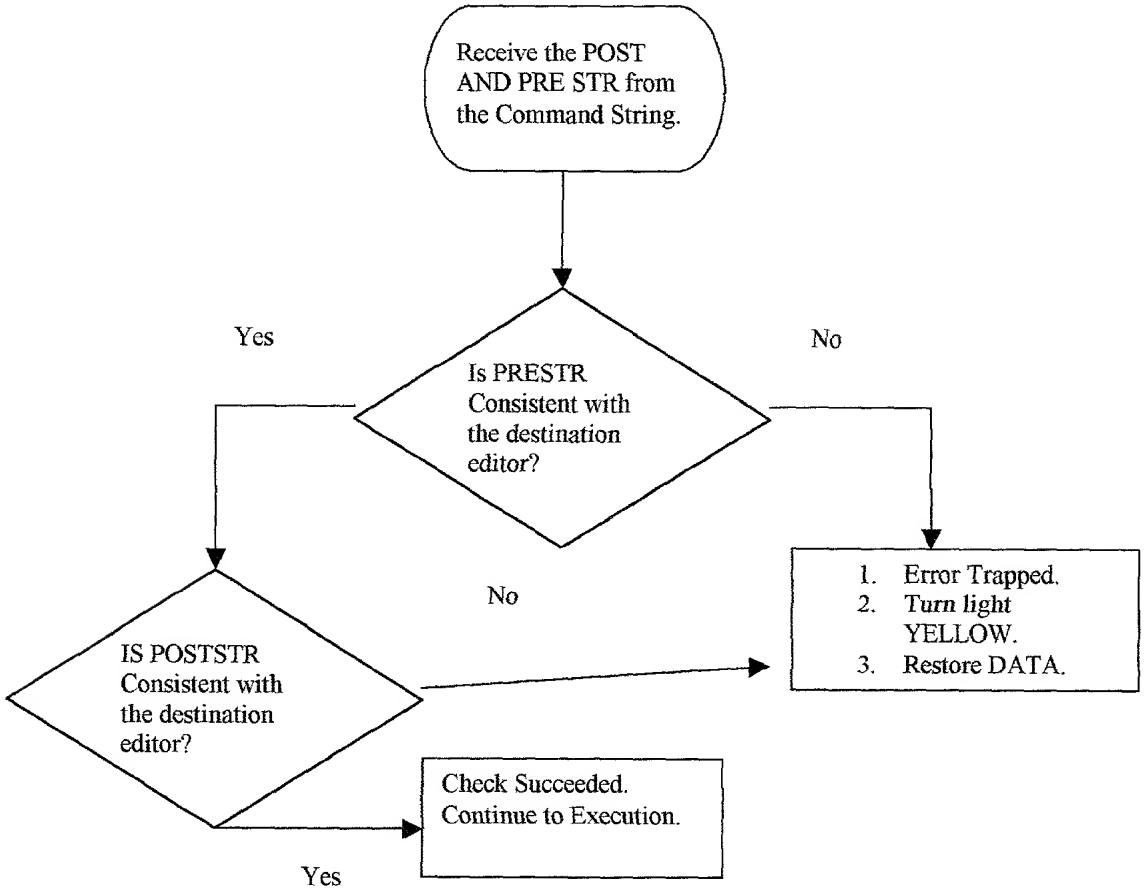
DATA CHECK STR

FLOW CHART No.7



DATA CHECK (Checking data while receiving).

FLOW CHART No.8



EXPLANATION OF FLOW CHART 1

This module handles the preparation of Command String based on User's Actions. As mentioned earlier, the editing can only be of 3 types: typing, formatting, OR clipboard operations. To handle each of these three, separate handling procedures were designed. They are presented under the heads ET, EF, ECP. Also, if the user does not any edit operation and clicks on Switch button, then also a Command String is generated (presented as ST).

Command String should also indicate whether the operation is destined from top → bottom or bottom → top editor. For this a flag "tb" (Top → Bottom) OR a "bt" (Bottom → Top) as the case may be, has to be attached. At the receiving end, first this is identified and then only the Command String is sent for execution at the corresponding destination editor.

ET (Editor Typing)

In this event, the SetA properties (namely, SelStart, SelLength, SelColor, SelFontSize, SelFontName, SelUnderline, SelBold, SelItalic, SelBullet, SelAlignment).

Following is a description of each of these properties:

SelStart: The current position of cursor.

Sellength: The length of text currently selected.

SelColor: The color code (a numeric value that is interpreted as a unique color).

SelFontSize: The font size of the selection of text.

SelFontName: The font name of the current selection.

SelUnderline: A flag that can be set to TRUE / FALSE. TRUE indicates that the selection of text is underlined. FALSE indicates the opposite.

SelBold: A flag that can be set to TRUE / FALSE. TRUE indicates that the selection of text is bold. FALSE indicates the opposite.

SelItalic: A flag that can be set to TRUE / FALSE. TRUE indicates that the selection of text is italicized. FALSE indicates the opposite.

SelBullet: A flag that can be set to TRUE / FALSE. TRUE indicates that the selection of text is bulleted. FALSE indicates the opposite.

SelAlignment: It indicates the Left / Center / Right alignment of the selected text.

Now, if there is no selection, then the Sellength will be ZERO. But still the system will get the values of each of these flags. The concept is that the properties at a particular

cursor position will always be those of the preceding cursor position unless the user has changed them at the current position (such actions one is recording, so the system always know the properties of the current cursor position).

Also, if the selection is a mixed one, i.e. if some part is underlined and some is not, then SelUnderline (and such properties) return NULL. So, system has to make a check on this before sending them to the buddy's editor for execution.

The character typed comes through the value of KEYASCII obtained as an argument to the event handler KEYPRESS of the editor. The KEYASCII value is a unique number that represents that character. For example the Capital letter "A" is equal to 65.

After having trapped what character has been typed, and all the SetA properties – giving special importance to the positional properties (SelStart and Sellength), system also pick up the PRESTR AND POSTSTR (the first5 and next5 characters at the point of edit – FLOW CHART NO. 7) These two strings are also merged with the information already gathered. Now, the actual Command String is prepared. It is further sent to the numbering shceme (DL-NUM) for processing.

EF (Editor Formatting)

This type of editing only changes the appearance of the text already present. It does not add or delete any text from the document. The normal of way of bringing out such changes on an editor is to select a part of text from the document (by highlighting), and perform the desired operation. This desired operation can be invoked wither by keyboard commands (like Ctrl B, Ctrl U, CTrl I) or Toolbar icons, or by Right Click menu option Font... (that launches a font dialog box).

In each of these operations, the main properties of importance are the cursor position of the selection start and the length of the selection. These are obtained by SelStart and SelLength.

Appropriate event handlers are incorporated in the system that identify a formatting operation. The first part in those event handlers is to make that change on the main editor, then to prepare a Command String so that the same operation happens on the buddy's editor.

For example if a bold operation has been performed, then the SelBold property of that selection is just negated (to revert the effect – i.e. bold becomes non bold and vice versa).

The changes property and the positional parameters are merged here and a Command String is prepared. Then this Command String is dispatched to the numbering mechanism for processing.

ECP (Editor Clipboard Operations)

Such actions deal with the clipboard. Out of these, only Cut and Paste change the document. Copy only takes some text to the computer's memory. Hence, the main actions here to be cascaded were Cut and Paste.

The clipboard actions are be default handled through the keyboard on any editor – i.e. Ctrl C (copy), Ctrl X (cut), and Ctrl V (paste). But programming was required to perform such actions through the toolbar icons and Right click menu options. For this, the ClipBoard object was utilized. For copying the text into the memory, the Clear (to clear the memory), and the SetText (to put text into memory) were used. To Cut, first the data was put into memory as explained in Copy and then the data was deleted from the document. For paste, the GetText method was used. It returns the text currently stored in the memory.

But similar actions just need to be reflected at the buddy editor, and not that the similar actions be performed on buddy memory. Hence, simulating actions were prepared for the clipboard actions performed on main editor.

For a Cut, just the Selstart and the Sellength of the Text cut was gathered and the Command String got prepared. For Paste action, the selstart, sellength, SetA properties and the pasted data (retrieved from the Clipboard.GetText method) was gathered and put into the Command String.

In case of the Paste data, this data is also compressed and packetized first and then merged into the command string.

After preparing the Command String, it is passed to the numbering scheme.

ST (Switch Transfer) (FLOW CHART NO. 4)

This occurs when the user clicks on the Switch button to transfer the switch of the document in the editor to the buddy's diagonal editor. This action triggers the following on within the system:

1. Making editor locked (locked=TRUE).
2. Making toolbar locked (locked=TRUE).
3. Making Red light visible and Green light invisible (visible = TRUE/FALSE).
4. Preparing a Command String that contains a flag TRUE to make it a Switch Command String.

After the preparation of the Command String is sent to the numbering mechanism.

EXPLANATION OF FLOW CHART NO. 2

This module handles the processing of editor data when a user joins a session. The basic criteria is that at any point of time, the contents of the diagonal set of editors should

remain the same. For this, it has to be made sure that when any user joins the session, the complete data present on top editor of the user is cascaded to the buddy's diagonal editor.

At this stage, if the buddy is present online (this status is always reflected by a STATUS property that contains "connected" or "disconnected"), then the whole data (RTF equivalent) of the top editor of each user is picked up, compressed, packetized, and then sent to the numbering scheme.

EXPLANATION OF FLOW CHART NO. 5

In this module each Command String ready to be sent out from the computer is tagged with a unique number. This number is a running sequence of numbers. This numbering makes sure that at the receiving end, a check can be done for any missing packet.

Once the number is tagged onto the Command String, then it is dispatch to the EC.

EXPLANATION OF THE FLOW CHART NO. 6 (EC – Editor Communication)

This communication is maintained by the intermediate Java programs. The processing of strings generated from end destined to go to the buddy end is explained in this flow chart.

Each Switch Notes session has a unique identification (switch_id). Each user also has a unique identification (strMname). Within each Switch_id, the system can have only authenticated users. The Java Server actually maintains a registry of such sessions and the users within them.

It is very important in this application that the data packets reach the end of the buddy who is in session only and not to any one else. For this, the actual physical identification of the buddy's computer on the internet (also know as the IP address) should be known. This IP address knowledge is extracted from the Java – Socket programming model. In this, a Client side and a Server side Socket for each client are made. For each socket, there is a socket ID and that in turn is related to the IP address of the user's computer. Hence, the main identifying factor in this model is the socket id of the computer.

To start the communication process, when the container of the ActiveX control loads (browser), the Client side hidden applet expects the strMname and the switch_id from the page as an input. This is passed into the applet through the PARAM tags (parameters). This set of information is carried to the Java Server. AT the Java Server, it is first checked if any other user is already present in the same session. IF this is the case, then an entry of this session id will already be present in the java server registry. If this is the case, then the buddy's status gets online for the newly entered user. A status information is sent to both the users confirming that both the users have been now connected. This means that now the communication can start between the two. But, if the case is that the user is the first to join the session, then a fresh entry of this session is made in the Java registry, and corresponding to his entry, the strMname of this user is also registered as present online.

From now on, if the Java server receives a packet destined to goto the buddy, then it just picks up the socket information of that buddy and sends it to the buddy.

In this flow chart, C1 and C2 stand for the Client1 and Client2.

EXPLANATION OF FLOW CHART NO. 3

This module handles the operations to be performed after the Command String has been received.

The first thing to check after receiving the Command String is whether there has been a data loss – i.e. whether any data packet has been lost on the way to the buddy. For this check, one number called previous_executed_string_number is always maintained. The number attached with the newly received Command String (i.e. packet) should be only and only ONE MORE THAN THIS previous_executed_string_number. If not, then it clearly indicates that either:

(1). One or more of the data packets have been lost on the way
(2). OR one packet has reached the buddy earlier than the first one.

Both of these cases are unacceptable. Hence, the instant this error is trapped, no further execution can happen on the editor (i.e. receiving editor). At this stage, the light of this editor is turned YELLOW indicating a data loss.

Next comes the data restoration. Asking the originating computer to send the Command Strings starting from this missing number again does this. Till the point this missing number packet is not received, the editor remains in data loss stage. This restorative mechanism is explained by the connector EXRS of the flow chart.

In case if there is no data loss, then the Command String is forwarded to the execution modules.

So, first it is checked as to what is the type of the Command String. It can either be an edit, a switch, or a Whole Data.

If it is an Edit, then the procedure EXED handles the Command String.

If it is a switch, then the procedure EXSW handles the Command String.

If it is whole data, then the procedure EXWD handles the Command String.

EXED (Execution for Edit)

Here, first a data check is run to ensure that the edit is going to be done at the right place.

DATA CHECK (FLOW CHART NO. 8):

In this checking scheme, the PRESTR and POSTSTR received as a part of the Command String is checked against the ones present already in the document. For example, if the words "switch notes" are present at position 10 in both the editors. The user types a character, say "4" at the space after the word switch. This means the new character has come at the position 16. Now, at the sending end, PRESTR would contain "switch" and the POSTSTR would contain " note" (a blank character is present before note").

The same two will be checked at the destination editor at the positions 10 to 15, then from 16 to 20. IF the both are found same, then only the editing will be allowed further. Else this will be notified as an error and again the a restorative action will take place – making light YELLOW, stopping any further executions.

Coming back to EXED.

Under this sub module, the following operations are carried out:

1. Set the Selstart to the one received.
2. Set Sellength to the one received.
3. Set the SetA properties to the one received.
4. Perform the corresponding edit operation – either by printing the received data OR by removing the pre-existing data.

EXSW (Execution for Switch)

This event is not an edit operation, hence no data check is needed here.

The following operations are carried out in this sub module:

1. EditControl.Locked = FALSE
2. Toolbar.Locked = FALSE
3. GreenLight.visible=TRUE
4. Redlight.visible = FALSE

EXWD (Execution for Whole Data)

This event handles the execution of Command String that carries the whole document data. This data will come in a sub packet form. Hence, first it has to be merged as one

packet at the receiving end. The number of sub packets are mentioned in the very first Command String that carries the information that a Whole Data Command String is on its way. The following are the steps carried out in this sub module.

1. Wait till all Sub packets arrive.

2. On arrival of last Sub packet, merge all into one packet.

3. Decompress this merged DATA.
4. Editor.TEXTRTF = Above Data. (Print the whole data into the editor – TEXTRTF stands for the RTF equivalent of the whole text).